# ΕΝΣΩΜΑΤΩΜΕΝΑ ΣΥΣΤΗΜΑΤΑ ΠΡΑΓΜΑΤΙΚΟΥ ΧΡΟΝΟΥ

**ΟΝ/ΕΠ : ΙΣΤΑΤΙΑΔΗΣ ΝΙΚΟΛΑΟΣ**

**EMAIL : nikoista@ece.auth.gr**

**ΑΕΜ : 9175**

## 1. ΒΑΣΙΚΑ ΣΤΟΙΧΕΙΑ ΜΕΛΕΤΗΣ ΚΑΙ ΔΙΕΞΑΓΩΓΗΣ ΣΥΜΠΕΡΑΣΜΑΤΩΝ

Στο συγκεκριμένο πείραμα έκανα μετρήσεις για λειτουργίες timer με περιόδους: 1sec, 0.1sec και 0.01sec στο Raspberry Pi με την εικόνα του λειτουργικού που σας έχει δοθεί .Το σκεπτικό μου πήγε ως εξής, πρέπει να κατασκεβάσω ένα πρόγραμμα που δεν κολλάει και αποφεύγει bugs σε συγκεριμένα σημεία ,δηλαδή ένα solid Pthread Producer Consumer πρόγραμμα σε γλώσσα προγραμματισμού C. Έτσι ένας Timer όπου είναι μια υπορουτίνα που αυτόματα εκτελεί κώδικα ανά τακτά χρονικά διαστήματα. 'Υστερα από αρκετές μέρες προγραμματισμού έφτασα σε ένα επιθυμιτό σημείο όπου το πρόγραμμα έδινε αρκετά ικανοποιητικά αποτελέσματα σε (ms) ,εκεί ξεκίνησα να κάνω μετρήσεις .

Τέλος ,τα στοιχεία του μηχανήματος μου Raspberry Pi 4 Model B με SoC που είναι BCM2711και Πυρήνας ARM Cortex-A72.΄Εχει 4 Cores με τον καθένα από τους τέσσερις πυρήνες , να τρέχει μόνο ένα νήμα κάθε φορά. Κάθε πυρήνας θα αλλάζει μεταξύ νημάτων (software) εκατοντάδες ή χιλιάδες φορές το δευτερόλεπτο, έτσι όπως οποιοδήποτε σύστημα που βασίζεται σε Intel ή AMD μπορεί να χειριστεί πολλά νήματα «ταυτόχρονα».

## 2. ΒΑΣΙΚΑ ΣΤΟΙΧΕΙΑ ΚΩΔΙΚΑ.

Ο κωδικάς μας απαρτήζεται από

5 βασικά στοιχεία:

**1) To struct timer.**

Όπου δημιουργεί αντικείμενα

timer και μια ουρά Queue όπου θα τα αποθηκεύση.

**2) To struct Queue.**

Δημιουργεί αντικείμενα Queue όπως και στην πρώτη εργασία σύμφωνα με το σκεπτικό fifo.

**3) To struct workFunction**

Δημιουργεί αντικείμενα workFunction και αυτά έχουν το καθένα μια σύναρτηση work και ένα όρισμα για την συνάρτηση arg.

**4) Την συνάρτηση Producer.**

Είναι μια συνάρτηση-thread που θα βάζει αντικείμενα τύπου workFunction στην ουρα.

**5) Την συνάρτηση Consumer.**

Είναι μια συνάρτηση-thread που θα βγάζει αντικείμενα τύπου workFunction στην ουρα.

## 3. ΑΝΑΛΥΣΗ ΚΩΔΙΚΑ

Καταρχάς η εργασία είχε πολλές ομοιότητες με την προιγούμενη .Είχαμε μερικές αλλαγές στον κώδικα λόγο του καινούργιου struct timer.

**1) ΓΙΑ ΚΑΘΕ ΑΠΟΜΟΝΟΜΕΝΗ ΕΚΤΕΛΕΣΗ**

Ξεκινάμε στην int main όπου εκεί αρχικοποιούμε μια ουρά fifo και ένα αντικείμενο τυπου timer τα κάνουμε Initialize,δημιουργούμε 2 threads consumer με την εξής μορφή

pthread_create (&con[i], NULL, consumer, fifo);

που θα βγάζουν αντικείμενα απο την ουρά και δίνουμε το timer αντικείμενο σαν όρισμα στην συνάρτηση start(t).Εκεί δημιουργούμε producer threads με την εξής μορφή

pthread_create(&t->UserData, NULL, **t->StartFnc**, t);

όπου πλεόν το

**t->StartFnc =** producer.

Στην συνέχεια μέσα στον producer

τοποθετούμε τα gettimeofday()

για να πάρουμε τους ζητούμενους χρόνους και δημιουργούμε workFunction αντικείμενα όπου τώρα για συνάρτηση work έχουν την TimerFnc. Αυτή υπολογίζει το

συνημίτονο μιας τυχαίας γωνίας με χρόνο εκτέλεσης να είναι ας πούμε 2usec . Έτσι δεν θα επιρεάσει την περιοδό μας όπου σύμφωνα με αυτή γίνεται η επαναλληπτική διαδικασία μέσα στην producer.Για συνθήκη τερματισμού έχω την εκτέλεση όλων τον Tasks που έχουμε και στην εκφώνηση της εργασίας (TasksToExecute). Από την άλλη μεριά έχουμε την consumer που είναι νήματα όπου μένουν αδρανή χωρίς να χρησιμοποιούν τη CPU μέχρι να μπει στην ουρά συνάρτηση προς εκτέλεση. Η ίδια ουρά και τα ίδια νήματα-εργάτες χρησιμοποιούνται από ένα ή περισσότερα αντικείμενα timer.

Η εκτέλεση είναι χωρίς διακοπή πέρα απο τυχόν διακοπές που επιβάλει το λειτουργικό σύστημα, ένας εργάτης αναλαμβάνει και τελειώνει μια εργασία πριν να αναζητήσει επόμενη από την ουρά. Τα νήματα-εργάτες δημιουργούνται από το χρήστη όταν δημιουργείται το πρώτο αντικείμενο timer και τερματίζονται όταν δεν υπάρχουν αντικείμενα timer.

Τέλος αποδεσμεύω μνήμη που χρησιμοποίησα στο πρόγραμμα και κάνω pthread_join τα threads μου καθώς και εμφανίζω στην οθώνη τον χρόνο (ελάχιστο, μέγιστο, μέσο όρο, διάμεσο, τυπική απόκλιση), που ξοδεύει η

A) producer για να βάλει μία κλήση στην ουρά.

B) consumer για να βγάλει μία κλήση από την ουρά.

## 2) ΠΑΡΑΛΛΗΛΗ ΕΚΤΕΛΕΣΗ

Ιδια υλοποίηση μόνο σχεδόν με μερικές μικρές διαφορές.

## 4 . TIME DRIFTING

Στα πειράματα έχουμε σταθερή μετατόπιση στον χρόνο (drifting). Για να την αντιμετωπίση της, κάθε νήμα producer ελέγχει τον χρόνο που πέρασε από την προηγούμενη φορά που εκτελέστηκε, ώστε να ρυθμίσει κατάλληλα την νέα διάρκεια - περίοδο όταν καλεί την συνάρτηση usleep, με απώτερο σκοπό την διόρθωση της χρονικής μετατόπισης.

Έτσι η gettimeofday() για να ελέγχει τον πραγματικό χρόνο ανάμεσα σε δύο διαδοχικές κλήσεις του νήματος producer και διαμορφώνει την νέα περίοδο.

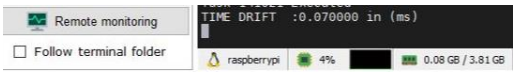
## 5 . ΑΝΑΛΥΣΗ ΣΥΜΠΕΡΙΦΟΡΑΣ ΤΟΥ ΚΩΔΙΚΑ ΣΤΟ ΜΗΧΑΝΗΜΑ RASBERY PI 4 MODEL B.

Βάζω QUEUESIZE = 100 έτσι ώστε να μην γεμίζει και να μην έχω queue is full που οδηγεί να καθιστερεί στον χρόνο και να χαλάει η περίοδος και μαζί το time drifting .

Επίσης έβαλα 2 thread consumer για καλύτερα αποτελέσματα συνολικά σε χρόνους.

Η χρήση της CPU που κάνει η διεργασία σας για κάθε πείραμα ήταν ανάμεσα στο

1% - 5%



## 6. ΣΥΜΠΕΡΑΣΜΑΤΑ ΚΑΙ ΠΑΡΑΤΗΡΗΣΕΙΣ ΠΑΝΩ ΣΤΙΣ ΜΕΤΡΗΣΕΙΣ

Η μετρήσεις έγιναν για Περιόδους:

**Α) 1 sec , Β) 0.1 sec , Γ) 0.01 sec**

## ΓΙΑ ΑΠΟΜΟΝΟΜΕΝΗ ΕΚΤΕΛΕΣΗ

Δύο χρόνοι έχουν νόημα:

- Πόσο διαφοροποιείται η κατάθεση της εκτέλεσης της εντολής από τη περίοδο που έχει προσδιοριστεί

- Πόσο χρόνο θα πάρει από τη στιγμή που θα κατατεθεί η εντολή εκτέλεσης μέχρι να ξεκινήσει η εκτέλεσης της.

**Α)** 1000ms = 1 sec ,

ο χρόνος εκτέλεσης του

worker.work=t-> TimerFcn

ήταν 2 (usec), χρόνος να μπει στην ουρά 1.456 (usec) και χρόνος δημιουργίας μέχρι να βγει από ουρά 25(usec) κατά μέσο όρο.

Το Time Drifting ήταν κατά μέσο όρο

85 (usec) ή 0.085 (msec) οπότε φτίαχναμε την περίοδο συνέχεια.

**Β)** 100ms = 0.1 sec ,

ο χρόνος εκτέλεσης του

worker.work=t-> TimerFcn

ήταν 1.8 (usec), χρόνος να μπει στην ουρά 1.453 (usec) και χρόνος δημιουργίας μέχρι να βγει από ουρά 23.5 (usec) κατά μέσο όρο.

Το Time Drifting ήταν κατά μέσο όρο

90 (usec) ή 0.090 (msec) οπότε φτίαχναμε την περίοδο συνέχεια.

**Γ)** 10ms = 0.01 sec ,

ο χρόνος εκτέλεσης του

worker.work=t-> TimerFcn

ήταν 1.85 (usec), χρόνος να μπει στην ουρά 1.450 (usec) και χρόνος δημιουργίας μέχρι να βγει από ουρά 23 (usec) κατά μέσο όρο.

Το Time Drifting ήταν κατά μέσο όρο

100 (usec) ή 0.100 (msec) οπότε φτίαχναμε την περίοδο συνέχεια.

**ΠΑΡΑΛΛΗΛΗ ΕΚΤΕΛΕΣΗ**

**Α) 1000ms = 1 sec ,**

ο χρόνος εκτέλεσης του

worker.work=t-> TimerFcn

ήταν 2 (usec), χρόνος να μπει στην ουρά 1.453 (usec) και χρόνος δημιουργίας μέχρι να βγει από ουρά 24(usec) κατά μέσο όρο.

Το Time Drifting ήταν κατά μέσο όρο

101 (usec) ή 0.101 (msec) οπότε φτίαχναμε την περίοδο συνέχεια.

**Β)** 100ms = 0.1 sec ,

ο χρόνος εκτέλεσης του

worker.work=t-> TimerFcn

ήταν 1.78 (usec), χρόνος να μπει στην ουρά 1.451 (usec) και χρόνος δημιουργίας μέχρι να βγει από ουρά 23.5 (usec) κατά μέσο όρο.

Το Time Drifting ήταν κατά μέσο όρο

103(usec) ή 0.103 (msec) οπότε φτίαχναμε την περίοδο συνέχεια.

**Γ) 10ms = 0.01 sec ,**

ο χρόνος εκτέλεσης του

worker.work=t-> TimerFcn

ήταν 1.86 (usec), χρόνος να μπει στην ουρά 1.457 (usec) και χρόνος δημιουργίας μέχρι να βγει από ουρά 23 (usec) κατά μέσο όρο.

Το Time Drifting ήταν κατά μέσο όρο

105 (usec) ή 0.105 (msec) οπότε φτίαχναμε την περίοδο συνέχεια.

# ΕΠΙΛΟΓΟΣ

Βλεπουμε πως η διαφορά τις παράλληλης λειτουργίας με την απομονομένη είναι πολύ μικρή για τους χρόνους queueAdd και queueDelete η της εκτέλεσης της TimerFnc.Το μόνο που ανεβαίνει είναι το Time Drifting όπου είναι δύσκολο να σταθεροποιηθεί και οδηγεί σε καταστροφή της περιοδικής διεργασίας.

Έτσι ένα online update τις περιόδου σε συνάρτηση με το Time Drifting είναι αναγκαία για την έγκυρη λειτουργία του προγράμματος και ταυτόχρονα της εφαρμογής που θέλουμε να υλοποιήσουμε. Επίσης παρατήρησα ότι τα ζευγάρια Producer/Consumer 1/2 3/2

οδηγούν σε καλύτερες επιδώσεις.

Τέλος ιδανικά, για λειτουργία πραγματικού χρόνου και έγκαιρη έναρξη QUEUESIZE 100 , 2 CONSUMER THREADS,T = 1s

TimeFnc time < 900 (ms)

# ΕΓΚΑΤΑΣΤΑΣΗ ΤΟΥ raspbian-espx , CROSS COMPLIE ΜΕ ΤΗΝ ΣΥΣΚΕΥΗ:

Κατέβασα το image που μας δώσατε και το έκανα εγκατάσταση το Raspberry Pi.Ακολούθησα τις οδηγίες για να επικοινωνήσω με την συσκευή , μέσο ethernet cable.Τέλος έκανα cross compile μέσο του eclipse όπου έβγαζα το c αρχείο σε binary μορφή . Έστειλα το binary αρχείο στο Raspberry Pi μέσο του command line καθώς οι δύο υπολογιστές επικοινωνούν και από εκεί το έκανα εκτέλεση
πχ ./Pthread_1 .